# A New Approach of Developing Compliance-Checking System

Xu Rong

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2004

## Abstract

Automate the process of building plan approval becomes a more and more urgent issue to current building industries. Existent code-compliance checking systems use the CAD data directly. While the CAD data can provide some low-level information, a lot of high-level semantic information required for code-compliance checking is not available directly. This thesis introduces a new approach of code-compliance checking. The major idea is to apply a pre-computation procedure to derive the semantic information from CAD data. The derived semantic information will be stored in a new building model called building checking object model, which will be used by code-compliance checking. In this way, the new approach of code-compliance checking provide a more effective and efficient solution. The checking request can be submitted by Internet and results will be visualized in 3D together with the 3D CAD models.

***Keywords:*** *STEP, Industry Foundation Classes, Code Compliance Checking, Code Checking Object Model, Computer Aided Architecture Engineering, 3D Modeling*

## *ACKNOWLEDGEMENTS*

*First of all, I would like to thank Dr. Huang Zhiyong who provided me invaluable advice and guidance throughout the course of writing this thesis.*

*I would also like to thank my family who understood me and supported my graduate studies.*

*Lastly, I would like to thank the school, which provided the opportunity and resources to make my thesis possible.*

## *Table of Contents*

# 1. Introduction

## 1.1. Background

Computer graphics have been successfully applied in architecture design. For example, geometric modeling and visualization provided alternatives for architects to model and evaluate 3D structures in ways more flexible than the traditional method of making a real scaled down model. There are more demands to extend visualization application into new areas. One of these new areas that we address in the thesis is the code-compliance checking of architectural plans.

Currently, building industries face more and more complicated regulations and codes of practice [7, 8] like other engineering industries. Every year, governments of countries spend huge amounts of manpower in validating building plans manually. The complexity and the changing nature of codes lead to delays in both the design and construction process. The designer must assess which codes are applicable to a given project as well as sort through potential ambiguity in the code provisions. The inspectors also must go through a similar process when doing approval. In addition, different inspectors may have different interpretations on a given section of codes. This inconsistency makes the approval process more difficult to be processed. Automating this process will speed up the building plan approval process and give both designer and permit-issue department a consistent framework to apply and check codes.

Recently, researches have been conduced to seek the automatic method for code compliance checking [1, 2, 5, 6]. Almost all of the research works are using Industry Foundation Classes (IFC) [4], a standard defined by International Alliance for Interoperability (IAI), as the basis for representing the building model information. In October 1997, the first building compliance checking system called BP Expert [10] was launched in Singapore. This system reads CAD data directly and checks them against those pre-loaded rules. Also in 1997, a client/server framework for online building code compliance checking is proposed by Charles S Han [1]. Within this framework, data from CAD system are extracted as IFC EXPRESS file and read by the code-checking server to produce checking results. The code-checking server is implemented in Java and all classes are strictly following the semantics of IFC semantics. In June 2001, the first commercial building compliance checking product called Solibri Model Checker (SMC) was announced at AEC (Architecture Engineering Construction) System Show in Chicago. SMC imports IFC R1.5.1 and IFC R2.0 product model files as its input data and do checking on the model.

While IFC is sufficient to achieve the interoperability of building information, its data nature is insufficient to support the code compliance checking. There is no provision in IFC to capture higher-level semantics of building elements while a lot of code-compliance checking requires high-level semantics of elements. As a result, most of the research works can only handle some simple checking, such as checking the fire rating of a wall. Neither BP

Expert nor SMC can handle complex checking such as calculate the travel distance from a space to a nearest exit staircase. The limitation of IFC prevents them from handling more complex checking.

## 1.2. Objective

Unlike other research works, this paper presents a better approach of developing code compliance checking system. To increase the capability of the system so that it will be able to handle more code compliance checking, a new object model called Code Checking Object Model (CCOM) is introduced. CCOM is actually a platform built up based on IFC object model and extended. CCOM object contains not only the low-level information that is already contained in IFC object model, but also contains high-level semantics such as object relationships, system networks and common checking logics, which cannot be found in IFC object model. While IFC model is still a building design model, CCOM becomes a building-checking model, which is more suitable to be used by code-compliance checking systems.

By using CCOM, new checking rules can be added into the code compliance checking system very easily because all the required high-level semantics is contained in CCOM object already. The consistency of basic checking logic will also be increased, as the common checking logics are included in CCOM, too. At the same time, some information in IFC that is useless for code-compliance checking will not be included in CCOM. Furthermore, by classifying the codes and regulations from different countries [5], compliance checking application can be customized to different code checking requirements of different countries [7, 8] by using CCOM.

Thus, the objective of this paper is to present a better approach of developing code-compliance checking system so that the system can be more compatible and consistent.

### 1.3. Paper outline

Chapter 2 briefly reviews the existing work in the area of code compliance checking.

Chapter 3 gives an overview of the code compliance checking system.

Chapter 4 describes the details of CCOM.

Chapter 5 analysis the algorithms used in CCOM.

Chapter 6 shows how checking is done based on checking rules and CCOM objects.

Chapter 7 introduces the web application module of the code compliance checking system.

Chapter 8 shows the implementation and testing results.

Chapter 9 gives the conclusion and lists the shortcoming of this system and possible improvement can be made in the future.

# 2. Related work

This chapter introduces some related works of code compliance checking system. We first introduce the Industry Foundation Classes (IFC), a widely used standard object model of building elements in CAD related software. Many code-compliance checking systems use IFC data as their inputs. Some important code-compliance checking systems include the BP-Expert developed in Singapore in 1997 [10], the client/server framework for online building code checking done by Charles S. Han [1] and the Solibri Model Checker (SMC) done by Solibri, Inc [11].

## 2.1. IFC (Industry Foundation Classes)

The idea of creating an integrated data model for achieving data sharing between multiple AEC software applications has been proposed by many researchers. For example, M. Sun's paper described an architectural CAD prototype originated from the Computer Models for the Building Industry in Europe (COMBINE) project [16]. However, only IFC becomes the common standard data format in AEC/FM domain.

In 1993, some of the major construction companies in the United States hosted a conference on the methods of using information technology in their industry. They wanted to work with each other's information regardless of the

kind of software they are using. Consisting of 12 companies involved in the AEC/FM (Architecture Engineering Construction / Facilities Management) industry, this group formed the International Alliance for Interoperability (IAI). IAI became a global organization in May 1996, a non-profit organization whose mission is to promote the development of the publications and specifications of the Industry Foundation Classes (IFC) [4] for AEC/FM information sharing through the construction life cycle. The IFC defines object-oriented data of buildings shared by all IFC compatible applications (Figure 2.1).

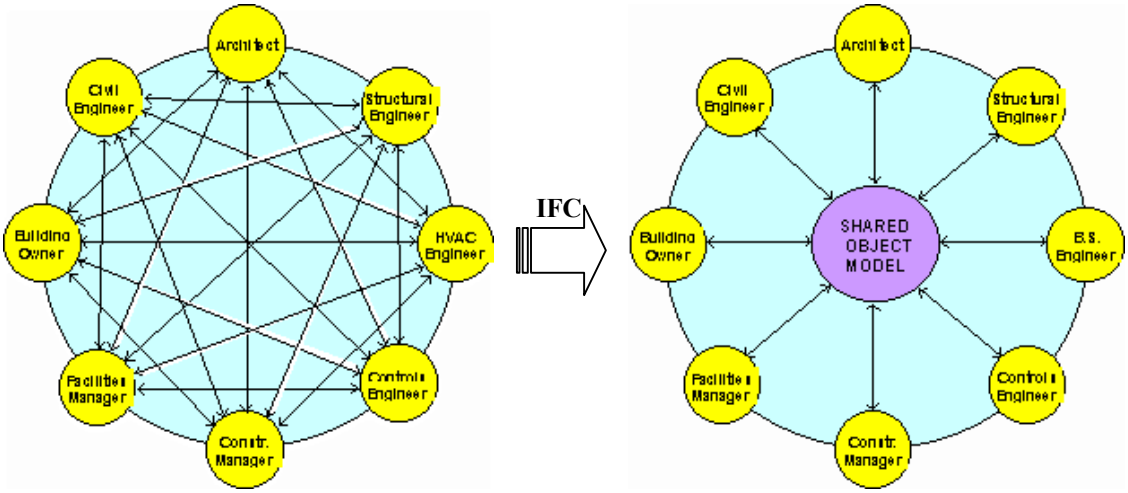



Figure 2.1: *IFC enables shared object models among AEC disciplines*

The intention of developing and using IFC is to have a common standard to specify how the 'things' that could occur in a constructed facility (the objects such as doors, walls, fans, etc. and abstract concepts such as space, organization, process etc.) should be represented. The specifications represent a data structure supporting the project model useful in sharing data across applications. Each

specification is called a 'class'. The class is used to describe a range of things that have common characteristics. For instance, every door has the characteristics of opening to allow entry to a space; every window has the characteristic of transparency so that it can be seen through. Door, window and fan are three examples of classes. A centrifugal fan object created in one application can be exchanged with and used in another IFC compliant application. This second application recognizes the centrifugal fan object, which reveals: a centrifugal fan, the amount of air to deliver against the amount of resistance, the operations, etc.

IFC specifies each building objects together with their characteristics. For example, IFC specifies a fan more than a simple collection of lines and geometric primitives of a fan. It knows that it is a fan and knows about the characteristics that make it one. Figure 2.2 shows how a fan is described in IFC.

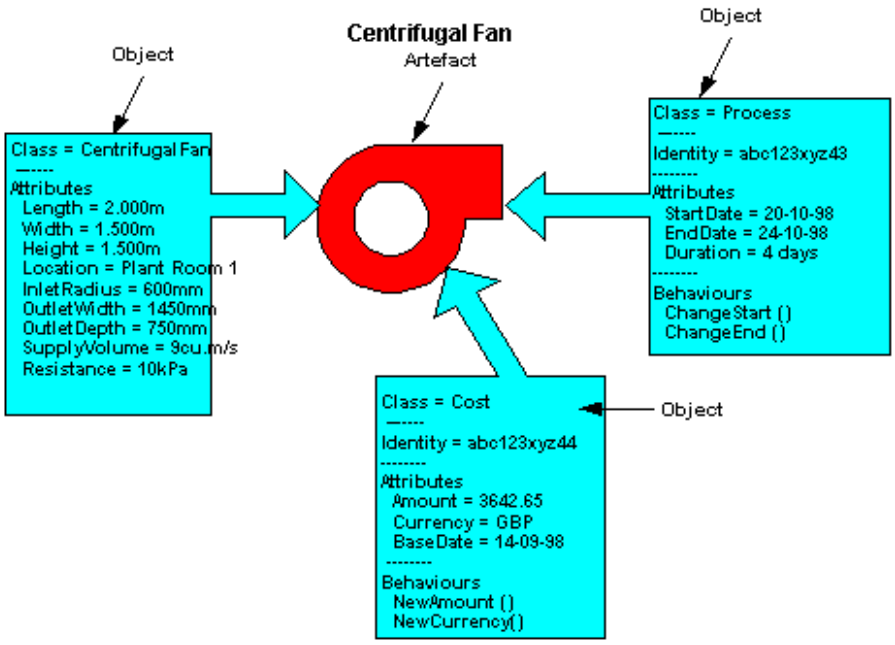



Figure 2.2: *Example of FAN in IFC model*

IFC enables interoperability among AEC/FM software applications. Software developers can use IFC to create applications that use universal AEC/FM objects based on the IFC specification. A centrifugal fan object created in one application can be exchanged with and used in another IFC compliant application. This second application recognizes the centrifugal fan object, which reveals:

*"I am a fan and I know that I am a centrifugal fan. I also know how much air I must deliver against what resistance offered by the ducted system I am connected to, the radius of my inlet connection and the length and width of my outlet connection. Additionally, I know what my operation is, what my geometry is, and so forth."*

The second application is able to understand these characteristics and add information to the object because it also uses the IFC. Applications that support IFC will allow members of a project team to share project data. This ensures that the data is consistent and coordinated. Furthermore, this shared data can continue to evolve after design, through construction, and occupation of the building. Information generated by the project design team will be available in intelligent, electronic format to the building construction team through their IFC compliant software and to building facilities managers through their IFC compliant software. Figure 2.3 shows the IFC information-sharing model.

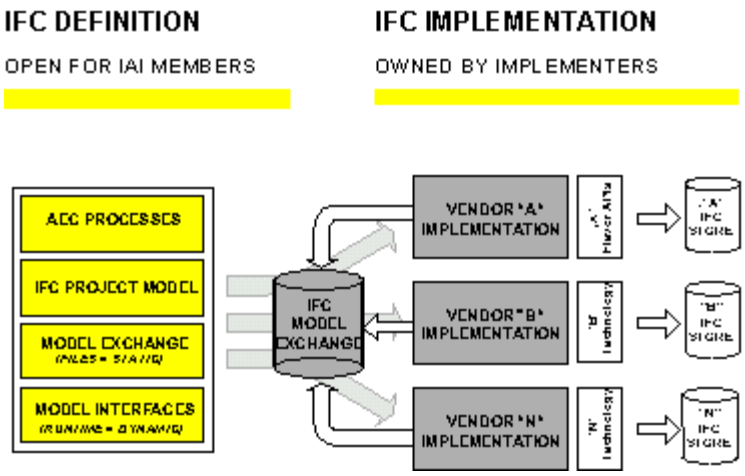



Figure 2.3: *IFC information sharing model*

## 2.2. BP-Expert

BP-Expert is a project lunched by the former Building Control Division (predecessor of the Building and Construction Authority) and the National Computer Board (predecessor of the Infocom Authority of Singapore) in 1997. Is it claimed to be intelligent software that combines the power of CAD technology with that of AI technology to automate the plan checking process. [10]. It allows CAD drawing to be directly inputted into the system. However, the BP-Expert can only supply AutoCAD drawing. Other CAD software's drawing cannot be used by BP-Expert as their formats are not the same as AutoCAD's. Another disadvantage of BP-Expert is that, the rules are coded into the software directly. BP-Expert can only check building plans against those rules that are already inside the software. No new rules can be added; no change to the original rules can be made. Figure 2.4 shows an overview of BP-Expert system.

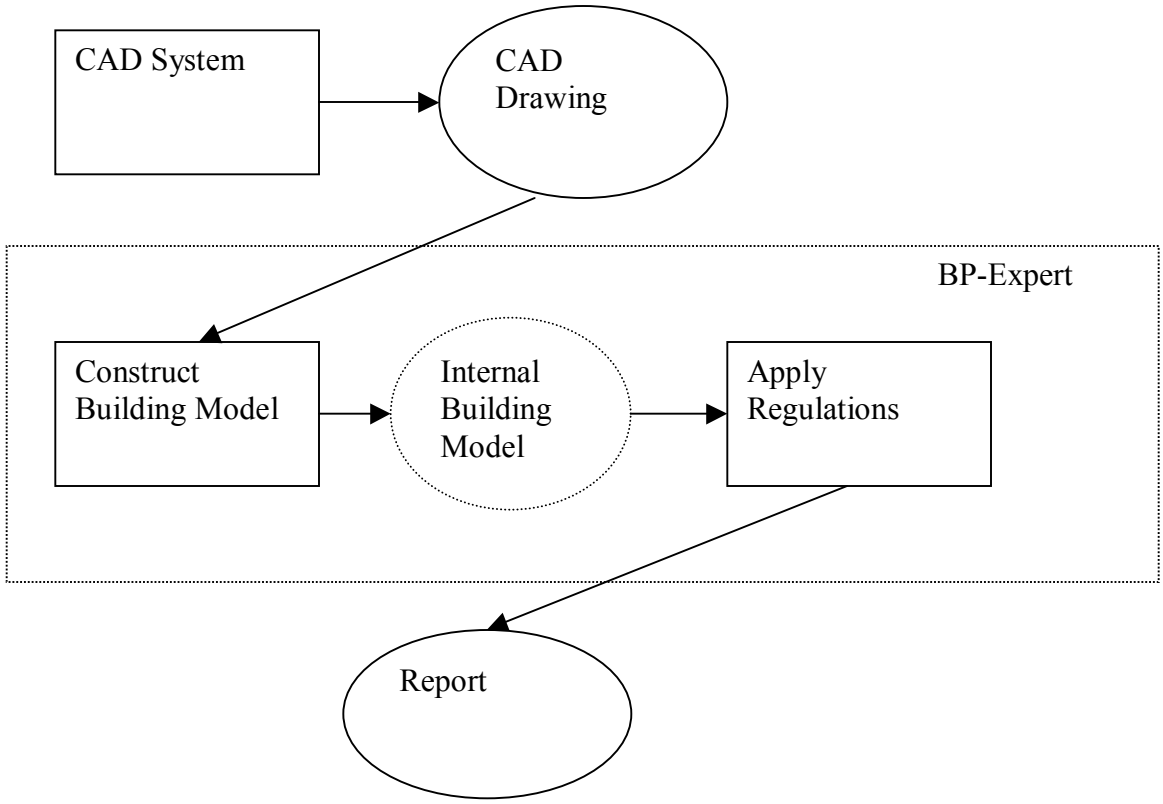



Figure 2.4: *BP-Expert*

## 2.3. Solibri Model Checker (SMC)

Solibri Model Checker (SMC) was announced at AEC Systems Show in Chicago in June 2001. SMC reads in a product model of a building and makes a check and analysis of typical design solutions [11]. One of the differences between SMC and BP-Expert is that SMC reads IFC R1.5.1 and IFC R2.0 product model as input. It makes SMC to be able to communicate with more than one type of CAD software including AutoCAD, ArchiCAD, etc, while BP-Expert can only

work with AutoCAD. However, the checking rules in SMC are still built in the software, which makes it very hard to change current rules and add in new rules. Figure 2.5 shows the components in SMC.

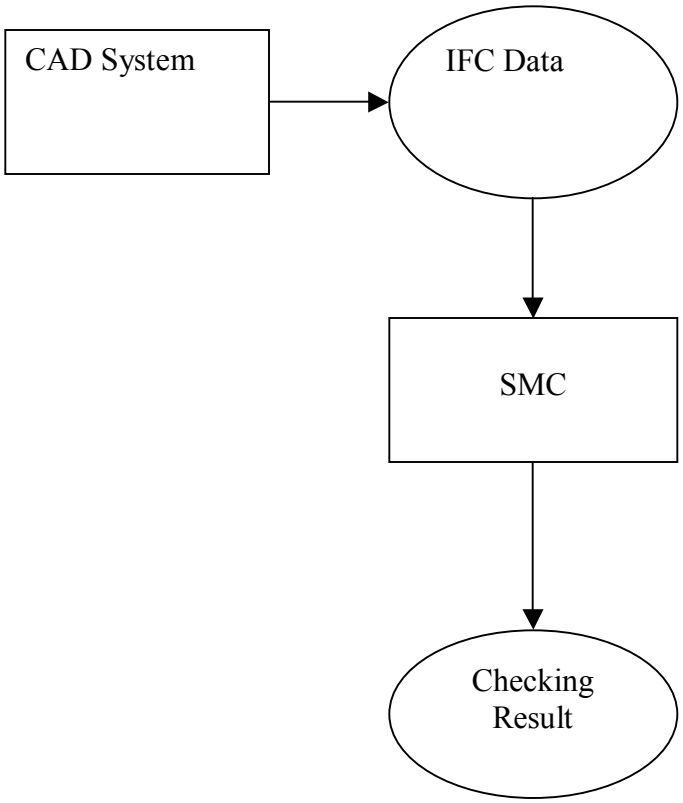



Figure 2.5: *SMC*

## 2.4. Han's code compliance checking system

Charles S. Han presents another client/server framework [1] of code compliance checking system in 1997. By separating the representation of building model and representation of code provision into different designed components,

Han's system turns to be much more flexible to add new rules. By using IFC object model to represent the building elements, Han's system is capable to work with the building design done by almost all kinds of CAD software. Figure 2.6 shows the components in Han's system.

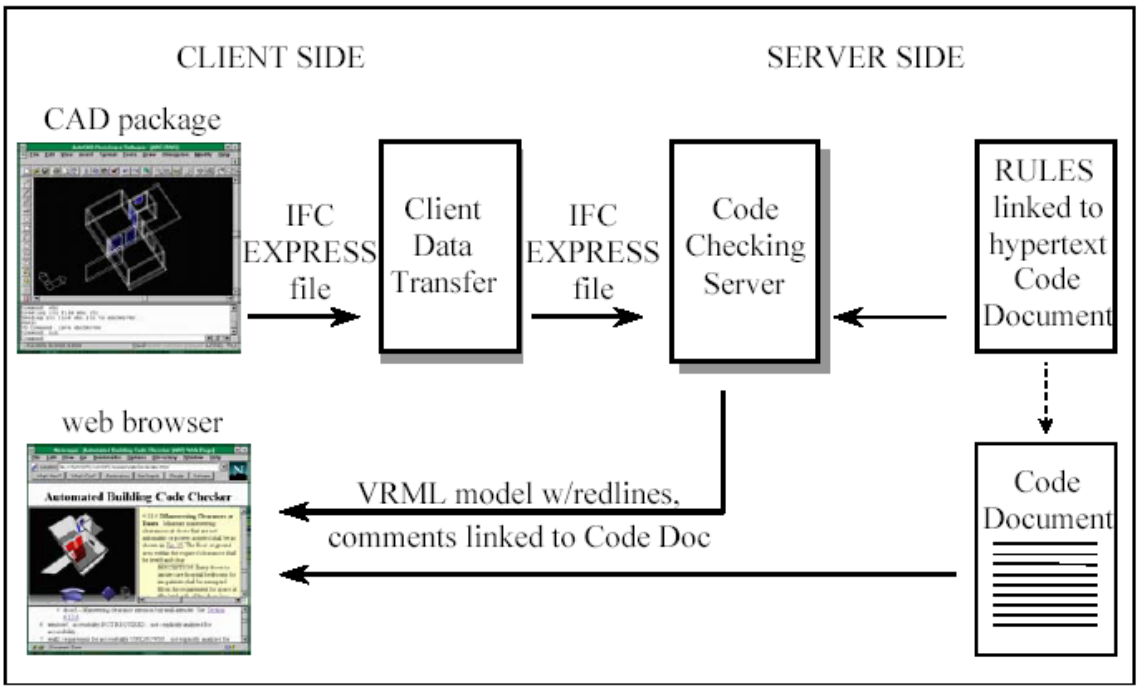



Figure 2.6: *Han's code compliance checking system*

## 2.5. 3D data structures

Solid-based data representation is commonly used to represent 3D data. Object-oriented data model that contains solid-based data and other data of the 3D object is also introduced by many researchers [12][15][17]. Most of these works are done for GIS systems.

Instead of GIS system, there have been several research efforts to develop object-oriented CAD system and object-oriented building models that contains the geometrical, functional, and behavioral relationships of building components [19][20][21][22]. However, all these works are not getting sufficient information for code-compliance checking system.

To make the code-compliance checking system more user-friendly, the checking result should be able to show and visualized on the 3D building model. Some similar works have been done by other researchers. Dogan proposed a relational database model to represent geometry and topology of 3D city data, which can be used to effectively visualize and query 3D city data [13]. Razdan's work [14] also shows that it is possible to derive semantical information from 3D geometry data. However, what Razdan did is only derive the semantical information of a single 3D object while what the code-compliance system has more interest on semantical information among all building elements.

# 3. System overview

The code compliance checking system that presented in this paper will separate representation of building model and representation of code provision into different components. IFC will still be used as input to the code compliance checking system so that the system can communicate with all kinds of CAD systems. However, unlike the previous research works, this code compliance checking system will not use IFC object model as representation of building model. Instead of using IFC object model as the representation of building model, a new object model called Code Checking Object Model (CCOM) will be used. There will be three modules in the system, which are CCOM module, Checking module and Application module.

CCOM module is the module to prepare data in CCOM model. CCOM data scheme will be generated based on semantics of CCOM objects together with the nature of codes and regulations through a scheme extraction process. With the well-defined CCOM data scheme, low-level information will be read from IFC data and high-level semantics will be generated based on low-level information (Detailed in Section 4). Figure 3.1 shows the major components and processes in CCOM module.

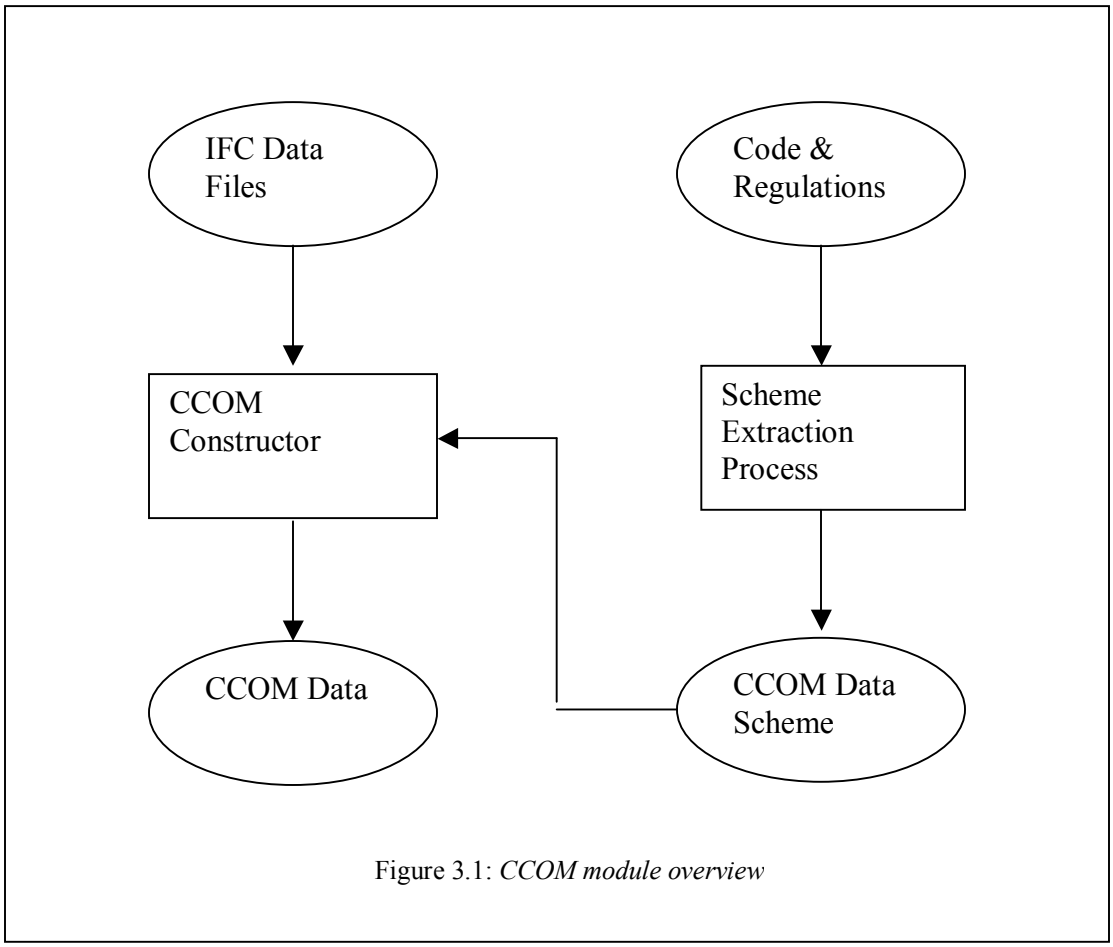



Figure 3.1: *CCOM module overview*

Checking module is the module to do code compliance checking. Checking rules are generated from codes and regulations through a rule generating process. Selected checking rules will be read into the code compliance-checking engine together with the required CCOM data to perform the checking and then checking result will be generated (Detailed in Section 5). Figure 3.2 shows the major components and processes in checking module.

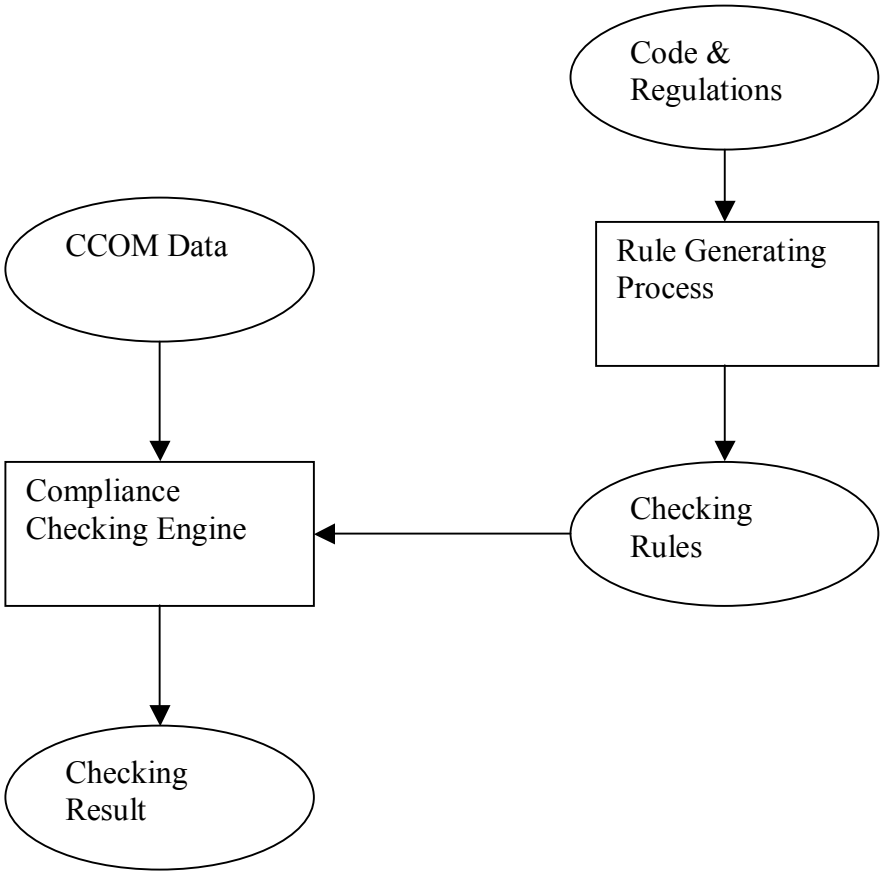



Figure 3.2: *Checking module overview*

Application module is the module dealing with human inter-actions. User can submit building/service plan and view it in 3D through web browser. When checking is completed, the result will be shown through viewer and formatted report will also be generated (Detailed in Section 6). Figure 3.3 shows the major components in application module.

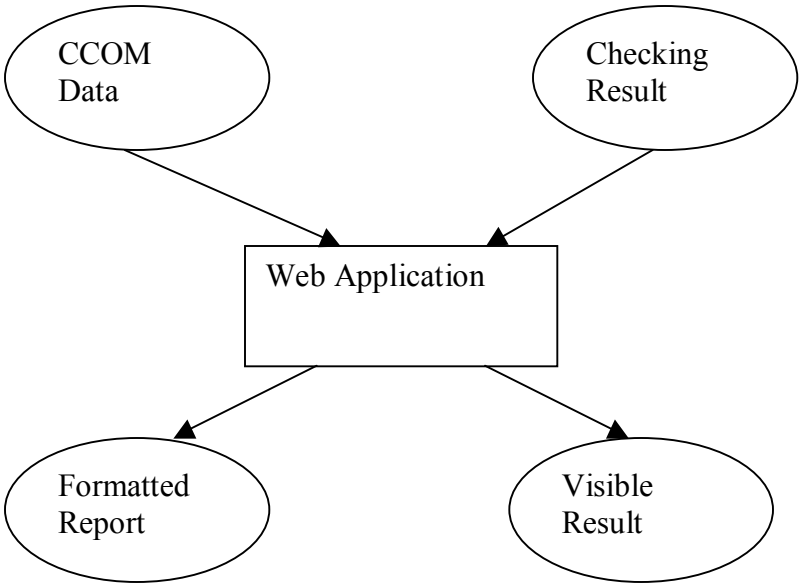



Figure 3.3: *Application module overview*

# 4. CCOM module

## 4.1. CCOM module overview

CCOM module is the module to prepare data in CCOM model. CCOM data scheme will be generated based on semantics of CCOM objects together with the nature of codes and regulations through a scheme extraction process. With the well-defined CCOM data scheme, low-level information will be read from IFC data and high-level semantics will be generated based on low-level information.

## 4.2. Motivation of CCOM

From the beginning, IFC is designed to achieve data interoperability. The intention of developing and using IFC is only to have a common standard to specify how the 'things' that could occur in a constructed facility (the objects such as doors, walls, fans, etc. and abstract concepts such as space, organization, process etc.) should be represented. The specifications represent a data structure supporting the project model useful in sharing data across applications. The data in IFC format is well structured towards building elements in isolation. We can get very detailed information about a particular building element, which is considered as low-level information, but little about the relationship between them and their behaviors, which is considered as high-level semantics.

Most of early research works are developed by directly using IFC object model as representation of building model. As IFC is lack of high-level semantics, it is very hard for complex checking to be performed.

For example, a kitchen and a corridor are merely spaces in the basic building information represented as Ifcspace in IFC model. In code checking the two spaces have very different meanings. A kitchen will require fire rated compartment in fire safety requirements, whereas a corridor must meet certain requirements related to shortest safe access to exit and protection in case of fire or must fulfill accessibility requirements for handicapped access. Although the spaces can be identified by names or types in IFC model, the characteristics specific to the higher-level semantics for code checking requirements are not captured in IFC, neither it is easy to be captured in design phase using CAD software.

A more complex case would be an apartment unit or exit staircase shaft. Common identification of such data in IFC would be the use of Ifczone, which is defined as a collection of spaces. An apartment unit will consist of a collection of spaces typically found in an apartment such as bedrooms, living room, toilets, kitchen, storeroom, etc. The same Ifczone is generally used to represent collection of spaces that form exit staircase shaft in vertical alignment. From IFC perspective, the two types of zones are identically represented. In code checking perspective, the two zones form two distinct “objects” with completely different

characteristics and requirements. Distance to the nearest exist staircase, most remote distance within the apartment to the exit door, area of the apartment unit are some of the very important characteristics required in an apartment unit. In an exit staircase shaft, vertical alignment of the spaces, staircase contained within the space - the width, number of steps, shape of it, exhaust system, pressure regulated requirements, discharge location, compartmentalization, property, location and swing direction of the door leading into exit staircase are important characteristics.

### 4.3. Definition of CCOM

Code Checking Object Model (CCOM) is essentially a 3D object model based on IFC that serves as a framework for building code compliance checking system. The Objective of CCOM is not to replace IFC to become a standard data format for data interoperability, but provide a platform for code compliance checking system development. It reads low-level information from IFC object and derives high-level semantics for code compliance checking system to use. In another word, CCOM is a middle level object model that is trying to fill the gap between code compliance checking requirement and IFC data capability. Figure 4.1 shows the relationship among IFC, CCOM and code compliance checking system.

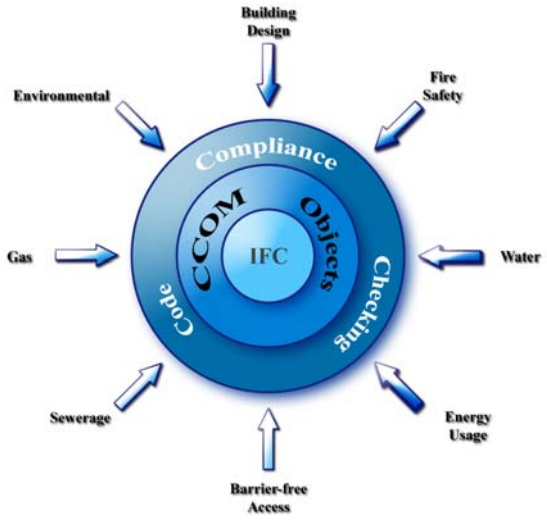



Figure 4.1: *CCOM objects bridge the gap between IFC and the code-compliance requirements.*

In CCOM model, each object is either a container or an element. A container contains an element and an aggregation of objects such as graphs, maps, trees, and lists representing the high level semantics. An element represents low-level information, similar to an IFC object, like static attributes and geometry representations of the objects. Figure 4.2 lists the formal definition of CCOM data model using the BNF.

**<CCOM_Object> ::= <CCOM_Container> | <CCOM_Element>**
**<CCOM_Container> ::= <CCOM_Element> <CCOM_Aggregation>**
**<CCOM_Aggregation> ::= <Aggregation_Structure> <Object_List>**
**< Aggregation_Structure> ::= <CCOM_Graph> | <CCOM_Map> | <CCOM_Tree> | <CCOM_List>**
**<Object_List> ::= <Empty> | <CCOM_Object> ; <Object_List>**
**<CCOM_Element> ::= <Attribute_List> | <Geometry>**
**<Attribute_List> ::= <Empty> | <Attribute> ; <Attribute_List>**
**<Attribute> ::= <Attribute_Name> <Attribute_Type> <Attribute_Value>**

Figure 4.2*: Formal definition CCOM using BNF*

For example, Space is an object. It is a container. The element part of a Space includes the attributes of the Space and also its geometry Representation. The attributes can be space name, space type space fire rating, etc. The geometry representation is usually an extruded solid. Of course it can also be represented as a Brep shape especially when the shape is not regular. The aggregation of the Space is a tree. It will contain all objects within the Space. The object in the aggregation can be either an element, for example, a flow terminal, or a container, for example, a space.

Figure 4.3 shows the example of CCOM_Object mentioned above.

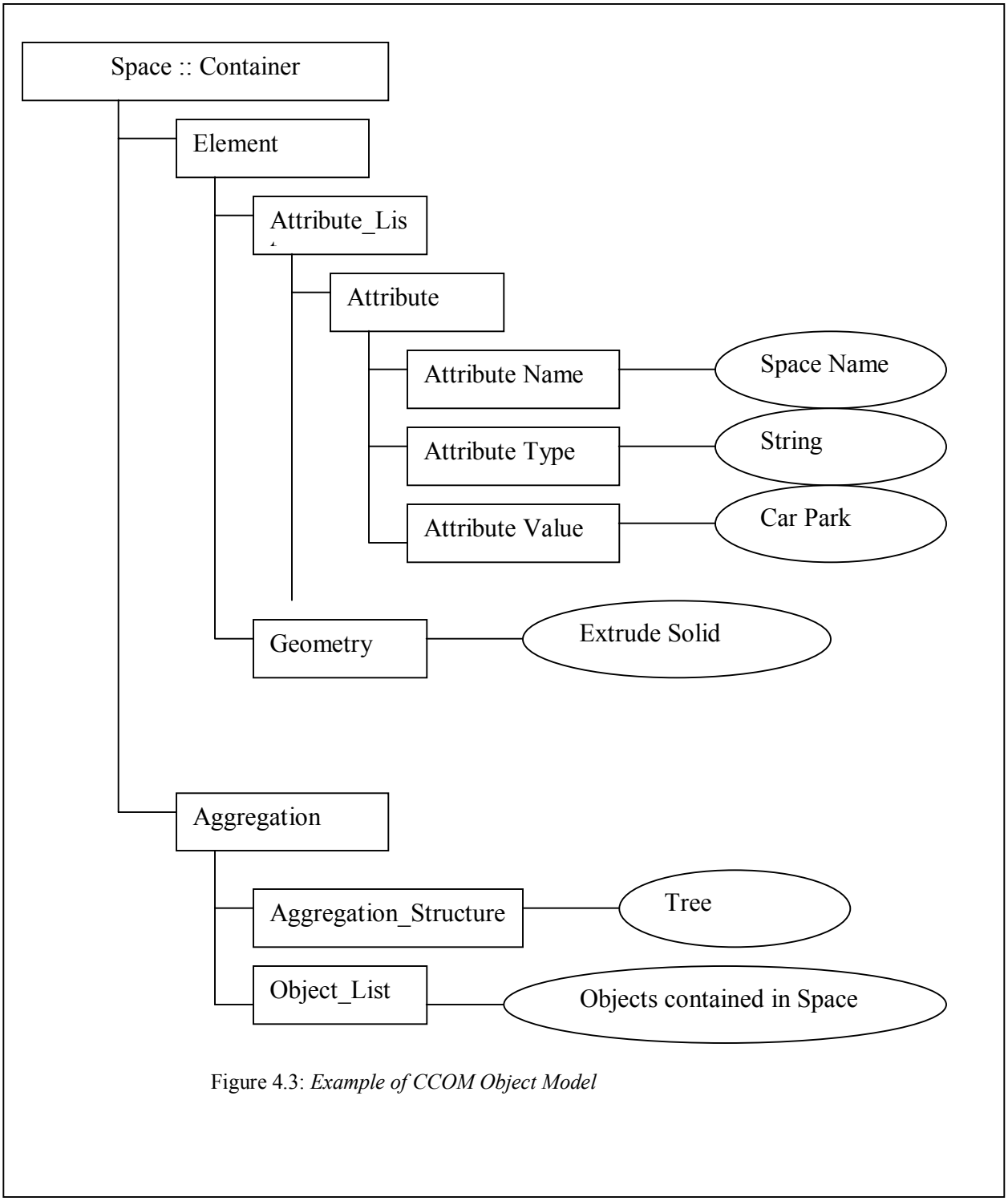



Figure 4.3: *Example of CCOM Object Model*

### 4.4. CCOM data construction

CCOM object data construction including follow steps:

- Define CCOM scheme from given Code & Regulations combine with the AEC domain knowledge. (Subsection 4.4.1)
- Get low-level information from IFC object directly based on the CCOM scheme. (Subsection 4.4.2)
- Derive high-level semantics based on existing low-level information and CCOM scheme. (Subsection 4.4.3)

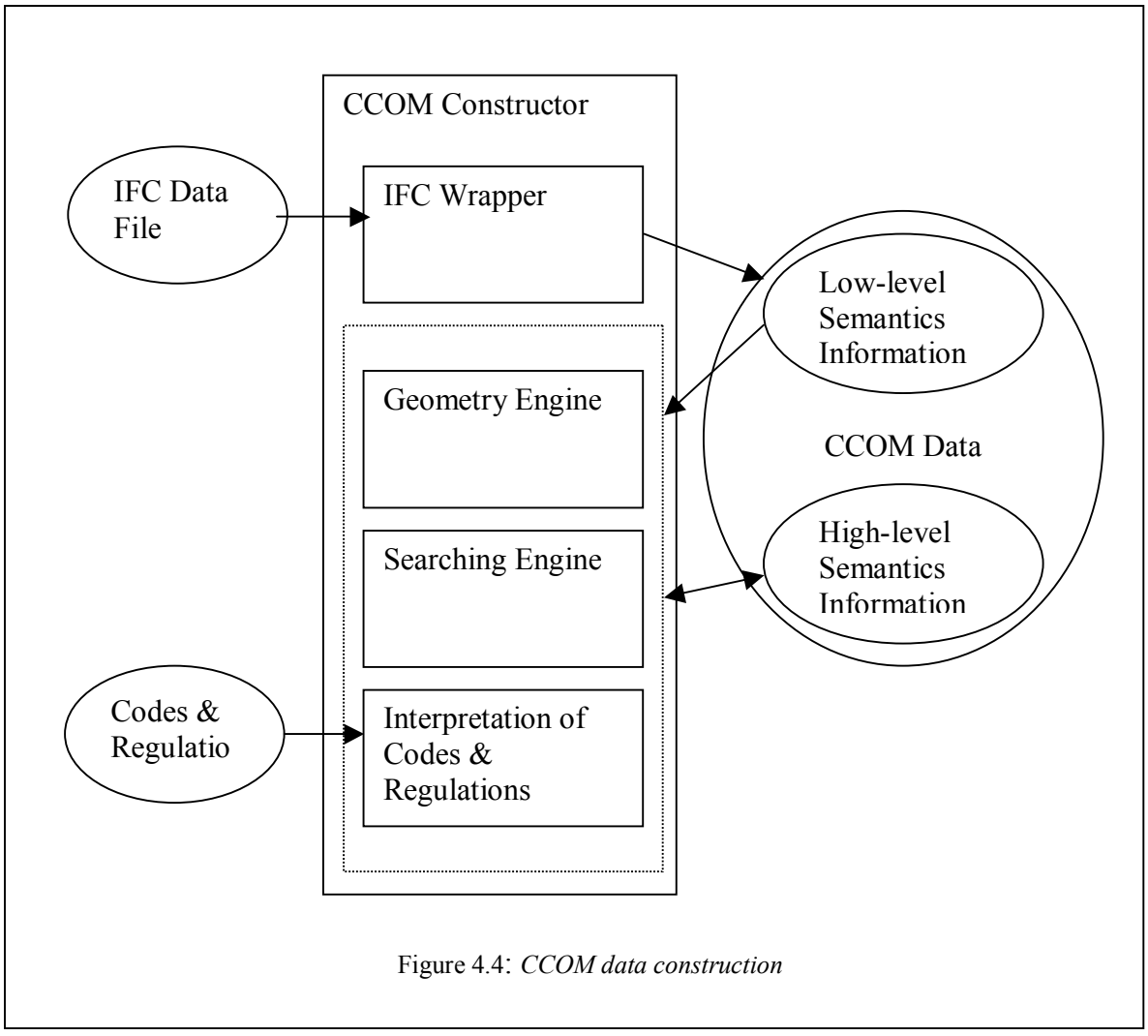



Figure 4.4: *CCOM data construction*

### 4.4.1. Define CCOM scheme from given Code & Regulations combine with the AEC domain knowledge.

Classes in IFC are defined based on the requirement of data sharing. Although it is also in AEC domain, compliance checking system will have different requirement on class architecture from the requirement of data sharing. With most of the classes similar to IFC classes, some IFC classes will be combined into same CCOM class, some IFC classes will be split into two or more CCOM classes, some IFC classes will be ignored in CCOM as they are not needed by code-compliance checking system.

As we have mentioned, an exit staircase shaft and an apartment unit, for instance, are represented by the same Ifczone object in IFC model but they both have different characteristics and requirements. Thus, the object 'Zone' in CCOM will be split into two subclasses called 'Exit Staircase Shaft' and 'Apartment Unit'. These two subclasses will have common attributes in 'Zone' while they will also have different high-level semantics in themselves.

Another example is the system for building service. The Ifcsystem object in IFC model is just a grouping for building elements. While in compliance checking for building service, different kind of system will have different requirements. 'Sprinkler System' is mainly dealing with provision and coverage of sprinklers, 'Ventilation System' is mainly dealing with ventilation control,

‘Water Supply System’ is mainly dealing with water supply, etc. Although some of the high-level semantics are similar among all systems, each of them will have their special behavior. As a result, Ifcsystem will also be split into several systems in CCOM model.

Along with the definition of CCOM_Object, whether the object is a CCOM_Container or a CCOM_Element will also be defined. ‘Exit Stair’, ‘Apartment Unit’ and ‘System’ are all defined as CCOM_Container. CCOM_Element includes ‘Flow Terminal’, ‘Pipe’, ‘Fan’, ‘Door’, ‘Window’, etc.

IFC already defines very detailed low-level information. However, those high-level semantics is still not well defined. High-level semantics needs to be defined carefully based on the interpretation of codes and regulations. For example, in Singapore’s code of practice chapter 29, section 2.3.2.2[4], breeching inlet is required to feed a tank less than 60 meters. The high-level semantics required is the height of the tank that the breeching inlet is feeding. The height of the tank is still not well defined as the height can be taken from top of the tank, bottom of the tank or any point of the tank. The height can also be referenced to the ground level of breeching inlet, the site level or sea level. From the interpretation of the code done by expert from AEC domain, we know that it should be from the center point of inlet pipe to the ground level of the nearest fire engine access way. Thus, high-level information required for this clause will

including inlet pipe position of a given tank, the tank feed by a given breeching inlet and the nearest fire engine access way of a given breeching inlet.

Another example is the 'System'. High-level semantics for all types of system will include the network graph of the system elements, the connectivity between elements, the shortest path between given elements within the system, etc. Each type of system will also have its own high-level semantics. 'Ventilation System' will have the total air change rate of the system, independency of the system, etc. 'Sprinkler System' will have the sprinkler coverage, anti-freezing schema of the system, etc. All these required information will be defined clearly according to codes and regulations.

With the definition of high-level semantics for each CCOM_Object, attributes of each CCOM_Object can also be defined. For a CCOM_Container, the Aggregation_Structure will be defined. For example, the Aggregation_Structure of a 'System' is defined as a Graph. The CCOM_Object to be contained inside the Object_List is also defined. Object_List of 'System' will contain all CCOM_Object assigned to the 'System', Object_List of 'Space' will contain all CCOM_Object within the 'Space', etc. Attribute_List of CCOM_Element is defined according to the requirement of each CCOM_Element and based on the corresponding IFC object.

### 4.4.2. Get low-level information from IFC object directly based on the CCOM scheme.

Low-level information can be read from IFC data directly based on the pre-defined CCOM scheme. To make this process more efficient, IFC data will first be imported to EDM database. CCOM constructor will use IFC wrapper to read required information from EDM database instead of the actual IFC file. Low-level information is stored in CCOM_Element.

The detailed process is like the following. For each Attribute_Name of a CCOM_Element, the value of corresponding IFC field of the corresponding IFC object will be read from via IFC Wrapper. The value will be cast to the Attribute_Value in the defined Attribute_Type and assigned to the CCOM_Element.

A more formal definition of this process is as following:

$Construct_{low\text{-}level}$ : (IFC field name, IFC field value) -> (Attribute_Name, Attribute_Type, Attribute_Value)

Figure 4.5 shows the process of reading low-level information from IFC data based on CCOM scheme.

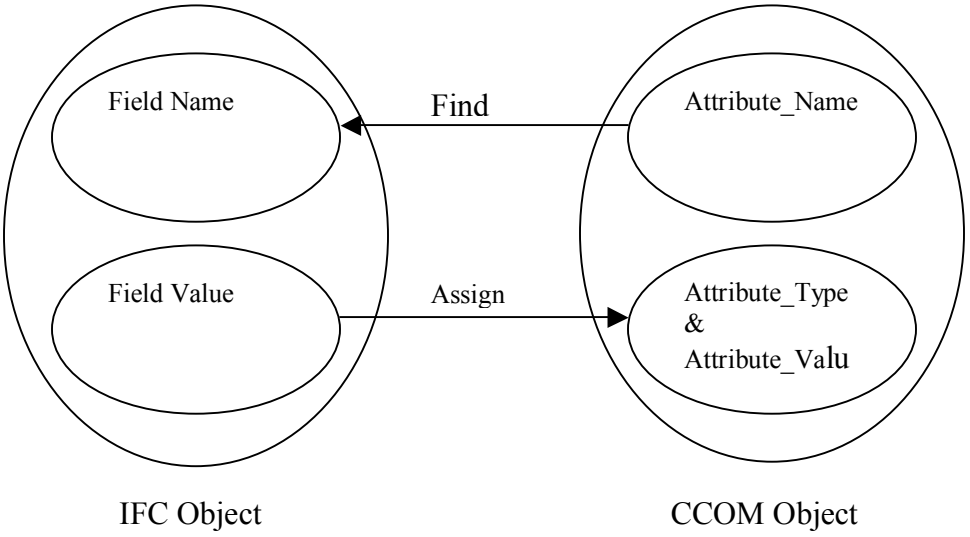



Figure 4.5: *Read low-level information from IFC data*

### 4.4.3. Derive high-level semantics based on existing low-level information and CCOM scheme

High-level semantics includes relationships among objects. The relationships can be either a physical relationship, such as containing, intersection, surrounding, etc, or a logical relationship, such as belongs to (a system), connected, at the top of, at the bottom of, etc. High-level semantics is derived based on the low-level information that read from IFC object. High-level semantics is contained in CCOM_Aggregation of CCOM_Container. To derive high-level semantics, geometry engine and searching engine are used.

When CCOM_Objects are contained in the same CCOM_Container, the relationship among CCOM_Objects that calculated by geometry engine and/or searching engine will be store in the structure of the CCOM_Aggregation. Different types of high-level semantics will be put in different types of aggregations based on the semantics nature. For example, the high-level semantics 'Containing' will be stored in a tree. Figure 4.6 is an example of the tree structure. From the tree, we can directly know whether an element is contained inside a container such as 'Building', 'Storey' or not.

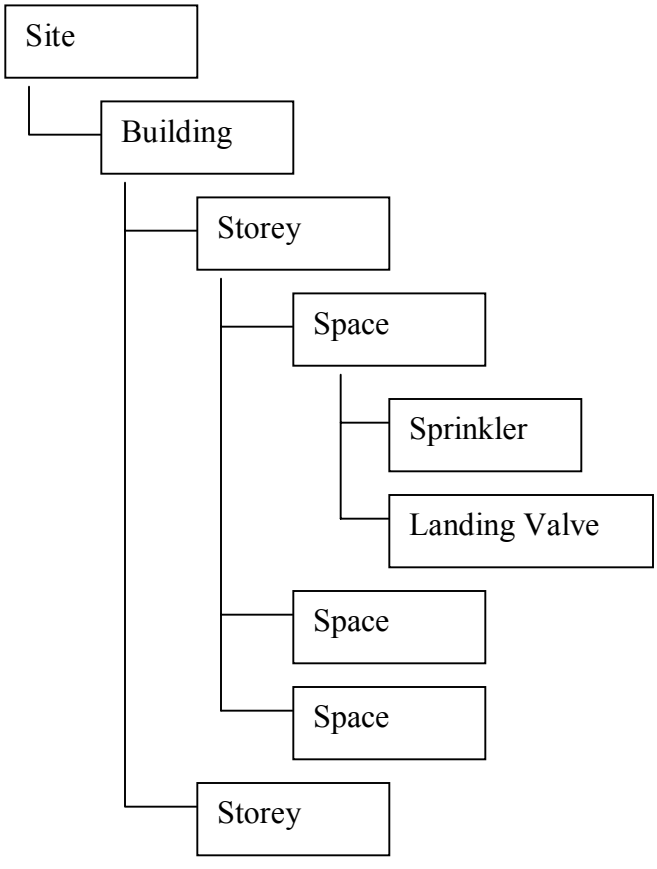



Figure 4.6: *An example of tree structure in CCOM_Container*

The high-level semantics 'Connectivity' will be stored in a graph. Figure 4.7 is an example of the graph structure. From the graph, we can get to know the connectivity relationships among elements such as "Hydrant A is connected to a PUB Main", "Hydrant A, B, C, D, E are forming a cycle", etc.

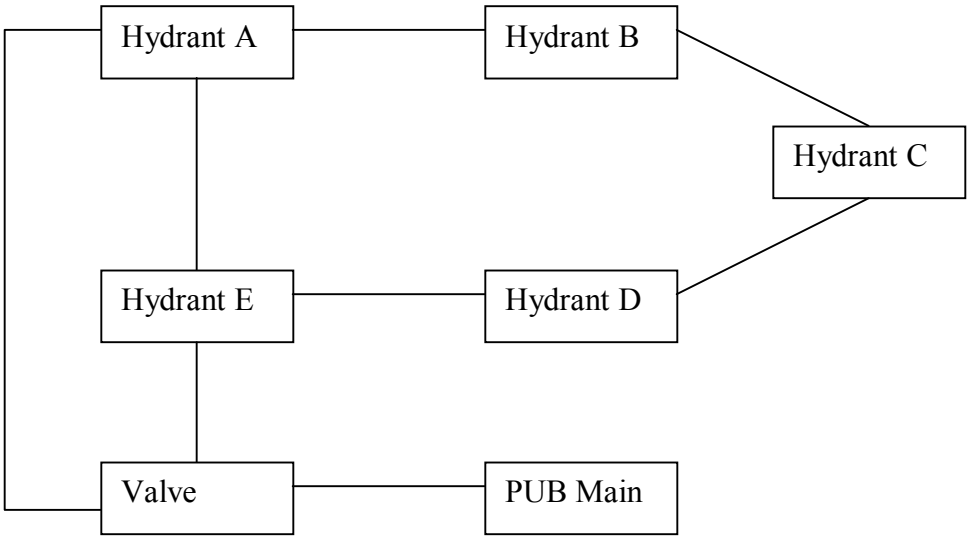



Figure 4.7: *An example of graph structure in CCOM_Container*

A more formal definition of this process is as following:

$Construct_{high\text{-}level}$ : (CCOM_Container, Object_List) -> (CCOM_Container)

Figure 4.8 shows the process of deriving high-level semantics.

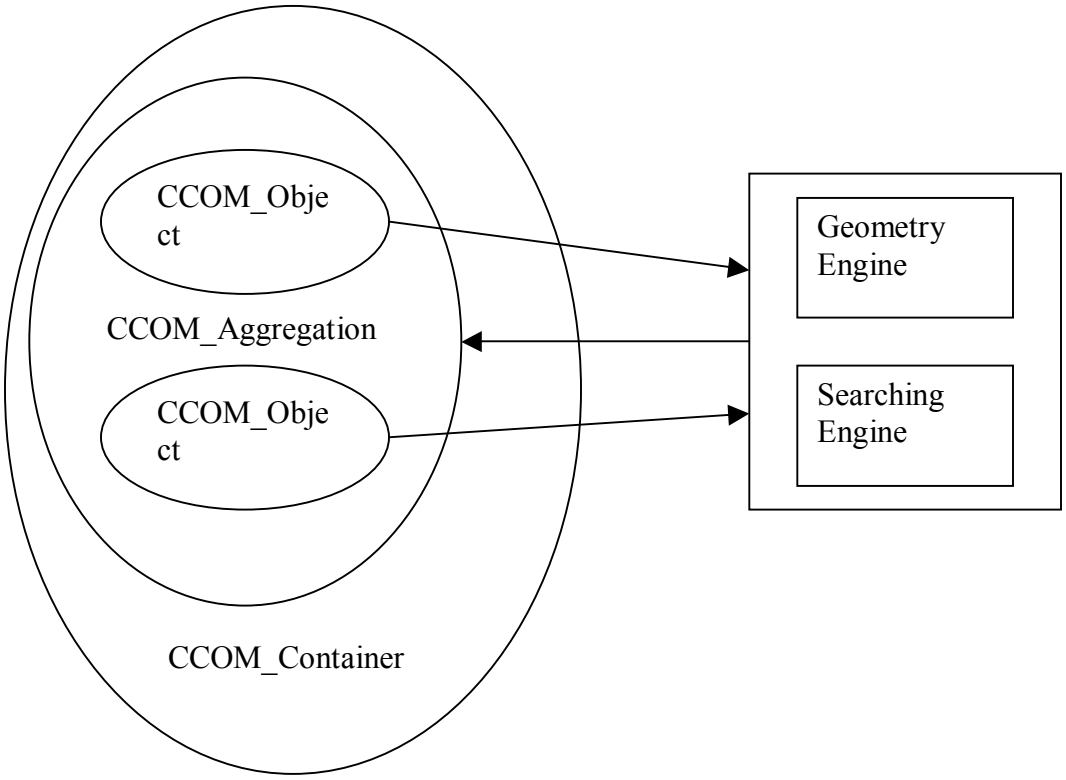



Figure 4.8: *Derive high-level semantics*

Using ‘Storey’ as an example. A ‘Storey’ is a CCOM_Container. It contains all the ‘Spaces’ of this ‘Storey’, as well as others CCOM_Object such as ‘Door’, ‘Wall’, etc. A very important high-level semantics is the travel distance from one point to another point. To get this high-level semantics, searching engine is used on the Aggregation_Structure of ‘Storey’. One of the Aggregation_Structure contained in the CCOM_Aggregation of ‘Storey’ is a graph (G). The nodes of G are all ‘Door’s contained in the Object_List of

CCOM_Aggregation. If two ‘Door’s are bounding a same ‘Space’, there will be an edge between these two ‘Door’s. The weight of the edge is the minimum travel distance between these two ‘Door’s within one ‘Space’. When trying to get the travel distance between two points in the ‘Storey’, the two points are also be inserted into G as two nodes. Edges will be built up between point and ‘Door’s that are bounding the ‘Space’ the point is contained. Weight of these edges is the minimum travel distance between the point and the ‘Door’. Shortest Path algorithm such as Dijkstra’s Algorithm will be used to find the shortest path between the given two points. Figure 4.9 shows the travel distance between a point to a ‘Door’. Figure 4.10 show the graph built up for calculating the travel distance. We can easily find from the graph that the shortest path between O and D is O-A-B-C-D and the shortest travel distance between O and D is 9 meters.

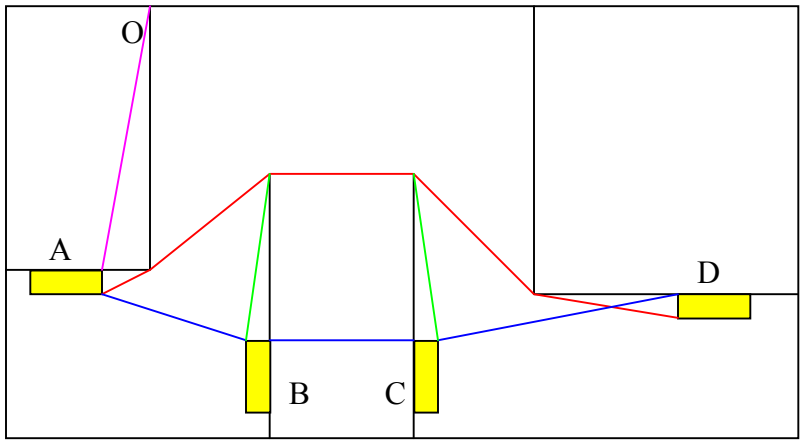



Figure 4.9: *Travel distance between a point to a ‘Door’*

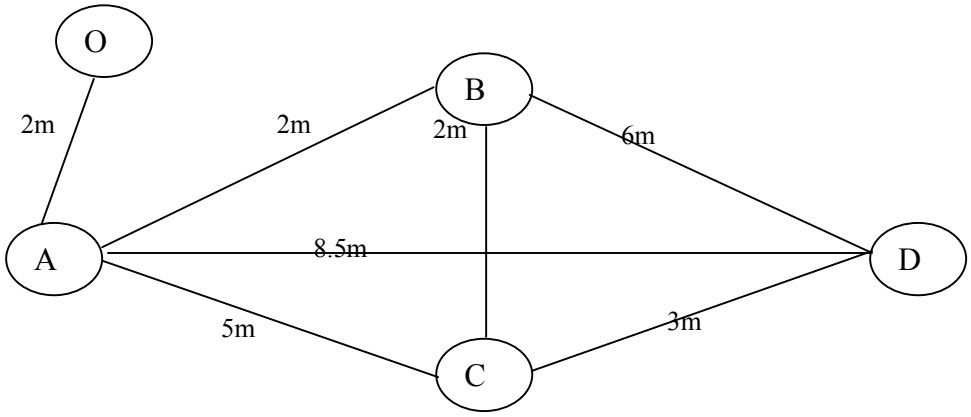



Figure 4.10: *Graph for travel distance*

Another example is 'Exit Staircase Shaft'. As 'Exit staircase shaft' is a vertically aligned circulation space that serves to deliver occupants to a safe external space in the event of catastrophe, the high-level semantics that are expected to be provided by CCOM object for code compliance checking will be finding the nearest discharge level for each storey. Figure 4.11 shows the example of how to calculate the nearest discharge level. Figure 4.12 shows calculation done by CCOM engine (including geometry engine and searching engine).

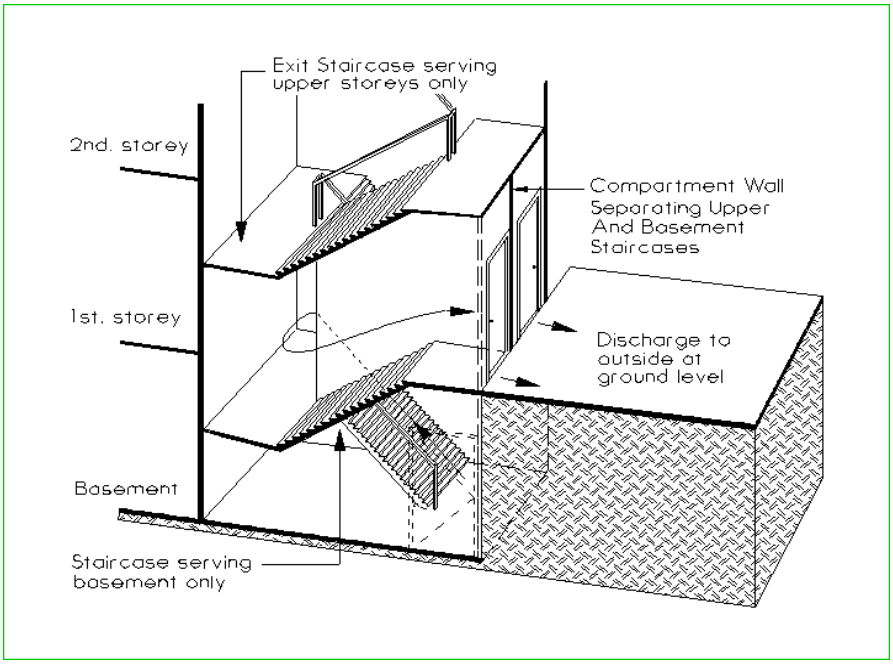



Figure 4.11: *Exit staircase shaft showing 1st storey as the discharge level for the basement.*

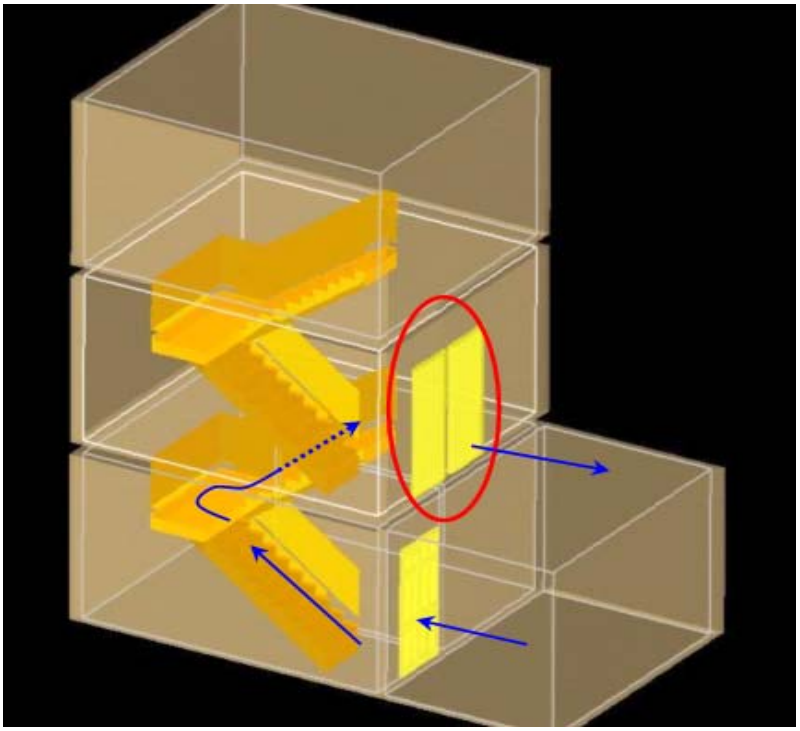


Figure 4.12: *A snapshot of calculation as done in CCOM for the nearest discharge level (shown with an ellipse) of the basement storey through the exit staircase shaft. The arrow shows the path from basement to the calculated discharge level.*

# 5. Algorithm analysis

In this chapter, we will discuss about the algorithms used in CCOM.

## 5.1 single-source shortest path algorithm

Single-source shortest path algorithms are mainly used when checking against two categories of codes and regulations, travel distance and building service system connectivity.

### 5.1.1 Breadth-first search (BFS)

Define a graph G = (V, E) and a distinguished source vertex s. A vertex's color is WHITE when it is unvisited, GRAY when it is visited and BLACK when all the adjacent vertexes are visited. Define $\pi(u)$ of vertex u to be the predecessor of u. Define Adj[u] of vertex u to be all connected vertexes of u. Define d[u] of vertex u to be the distance between u and s.

BFS (G, s)

```
1     for each vertex u ∈ V[G] – {s}
2           do color[u] ← WHITE
3              d[u] ← ∞
4              π[u] ← NIL
5     color[s] ← GRAY
6     d[s] ← 0
7     π[s] ← NIL
8     Q ← {s}
9     while Q ≠ ϕ
10          do u ← head[Q]
11             for each v ∈ Adj[u]
12                do if color[v] = WHITE
13                      then color[v] ← GRAY
14                           d[v] ← d[u]+1
15                           π[v] ← u
```

```
16                                          ENQUEUE(Q, v)
17              DEQUEUE(Q)
18              color[u] ← BLACK
```

In CCOM, BFS algorithm is mainly used to find the shortest path between two elements. However, a modified BFS algorithm can also be used to find the nearest connected element which full-fit the requirement. A Boolean function IsRequired(u) of vertex u returns true if u full-fit the requirement and false if not. The BFS will be modified to break out once a required vertex is found. The total running time of BFS is O(V+E) where V is the size of vertexes and E is the size of edges.

### 5.1.2 DFS

Define d[u] of vertex u to be discovery time and f[u] to be finishing time.

```
DFS (G)

1       for each vertex u ∈ V[G]
2               do color[u] ← WHITE
3                       π[u] ← NIL
4       time ← 0
5       for each vertex u ∈ V[G]
6               do if color[u] = WHITE
7                       then DFS-Visit(u)

DFS-Visit (u)
1       color[u] ← GRAY
2       d[u] ← time ← time + 1
3       for each v ∈ Adj[u]
4               do if color[v] = WHITE
5                       then π[v] ← u
6                               DFS-Visit(v)
7       color[u] ← BLACK
8       f[u] ← time ← time + 1
```

In CCOM, DFS algorithm is mainly used to form the connection forest of all building service elements and detect the connected components. DFS algorithm is also used to check whether there is any loop within the building service systems. The total running time of DFS is $\theta(V+E)$.

# 6. Checking module on top of CCOM

Checking module is the module to perform the checking rules on top of CCOM data. The checking rules are formed based on the interpretation of codes and regulations.

CCOM_Objects are selected based on the checking rules and information of these CCOM_Objects will be retrieved for calculating the final checking result.

We define the checking engine as following:

Checking : (Checking Rules, CCOM_Object) -> (Checking Result)

For example, in Code of Practice from FSB, Chapter 29, clause 2.2.2, each space should be within 38 meters from landing valves. The clause is described as following:

Code of Practice (s/n 3)
Clause 2.2.2 Number of Rising Main
Number of Rising Mains
The number and distribution of rising mains shall be such that:
a) All parts of the floor not more than 24m above the ground level is within 38m from landing valves. The distance should be measured along a route suitable to hose lines, including distance any up or down stairway.

Figure 6.1: *Example of Code & Regulations.*

The interpretation of this clause will be as following:

This clause check the rising in the building where

1. The floor is not more than 24m from the ground level.
2. Each space shall be within 38m from at least one landing valve through a suitable route.

Figure 6.2: *Example of interpretation of Code & Regulations.*

Based on the interpretation, Modeler in AEC domain will define the following checking rule as described in figure 6.3:

1. Get all storey from the building
2. For each storey not more than 24m
   - 2.1. Get all spaces from the storey
   - 2.2. Get all landing valve from storey
   - 2.3. For each space
     - 2.3.1. Calculate the remote point in this space from its door(s)
     - 2.3.2. For each landing valve in the storey
       - 2.3.2.1. Calculate the travel distance from the landing valve to the remote point
       - 2.3.2.2. If the travel distance is less than 38m, space pass.
   - 2.4. Space fail

Figure 6.3: *Example of Checking Rules.*

Based on the checking rule listed in figure 6.4, the checking engine will perform the checking on the corresponding CCOM objects as following:

1. Set of Storey := Building->GetAllStorey
2. For each Storey
   - 2.1. Set of Space := Storey->GetAllSpace
   - 2.2. Set of Landing_Valve = Storey->GetAllLandingValve
   - 2.3. For each Space
     - 2.3.1. Remote_Point := Space->GetRemotePoint
     - 2.3.2. For each Landing_Valve
       - 2.3.2.1. TravelDistance := Storey->CalculateTravelDistance(Remote_Point, Landing_Valve)
       - 2.3.2.2. If TravelDistance < 38m, then Space pass, goto next Space
   - 2.4. Space fail

Figure 6.4: *Checking based on CCOM objects and checking rules.*

In figure 25, CCOM_Object including Building, Storey, Space, Landing_Valve, Remote_Point. The high-level semantics of each CCOM_Object used are listed in figure 6.5. Notice that all of the high-level semantics cannot be got from IFC data directly.

| **CCOM_Object** | **High-level semantics** |
|---|---|
| *Building* | *GetAllStorey* |
| *Storey* | *GetAllSpace*<br>*GetAllLandingValve*<br>*CalculateTravelDistance* |
| *Space* | *GetRemotePoint* |

Figure 6.5: *high-level semantics used in the checking example.*

The checking result of above checking is shown in figure 6.6. The highlighted space is failed because it is too far away from a landing valve. Other spaces passed the clause.

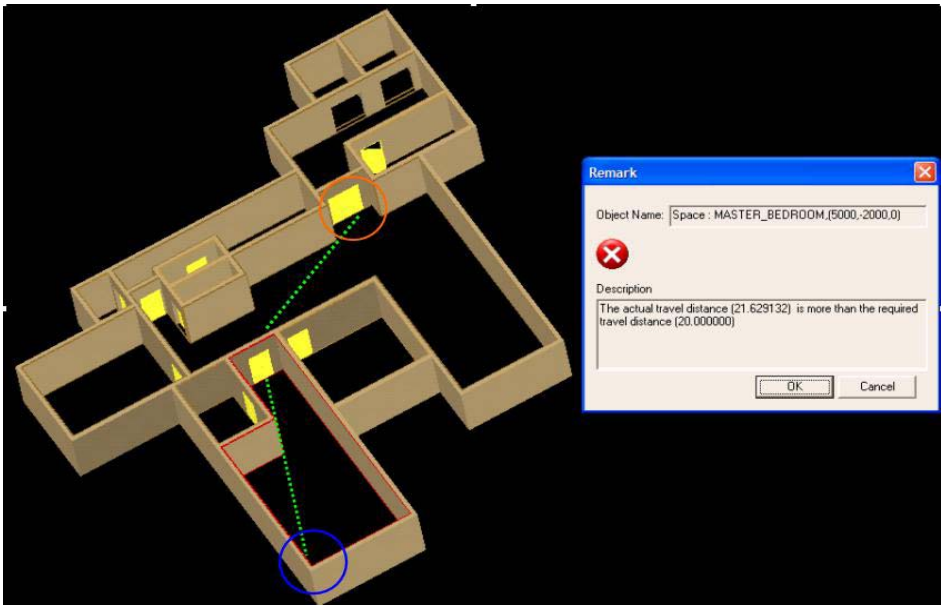



Figure 6.6: *checking result.*

# 7. Application module

Application module is the module dealing with human inter-actions. User can submit building/service plan and view it in 3D through web browser. When checking is completed, the result will be shown through viewer and formatted report will also be generated.

The building design documents will be submitted to the system via web browser.

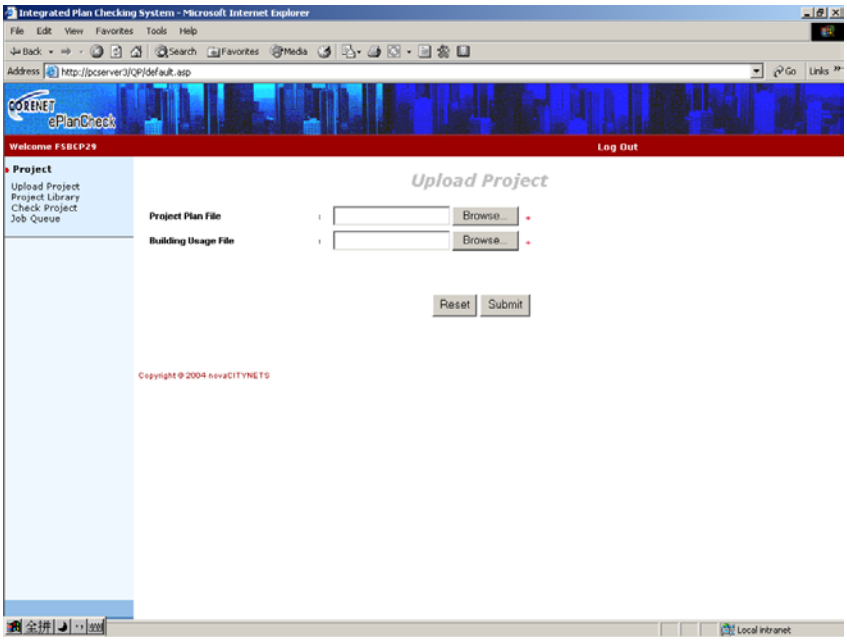



Figure 7.1: *submit building plan.*

The building design will be able to be viewed via web browser.

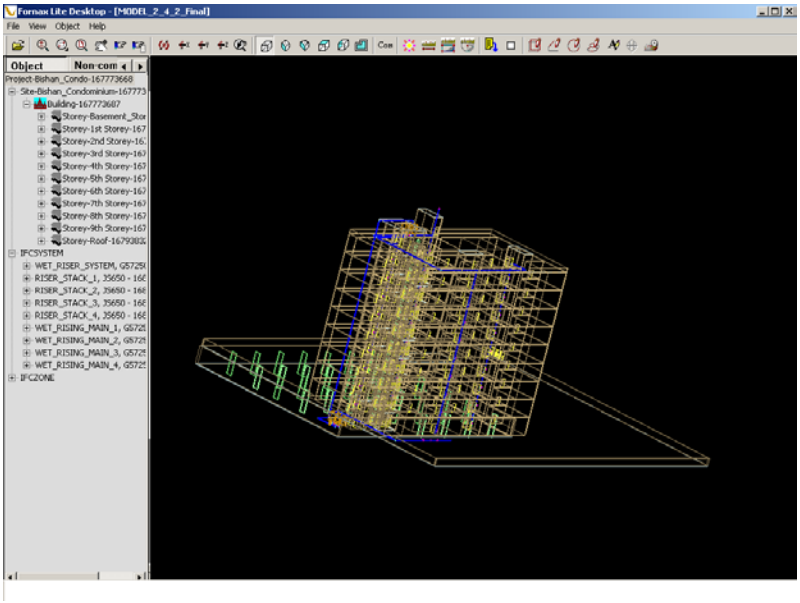


Figure 7.2: *check against selected code*

By choosing the code or regulation to be performed, the system will check the building against the code or regulation. Checking result will be displayed and building elements that against the rules will be highlighted in the viewer.

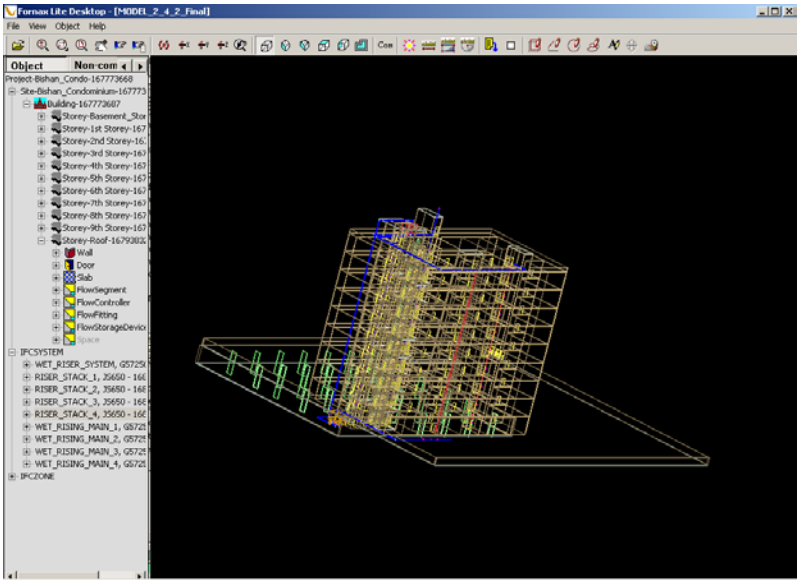


Figure 7.3: *highlight the result.*

# 8. Implementation and results

CCOM module and checking module are implemented in C++ and ACIS is included as a geometry engine. Simple Object Access Protocol (SOAP) is used to transfer object model between client and server. The viewer is implemented in C++ with OpenGL support. Data streaming technology is used to transfer building model between client and server. The web application part of the system is implemented in ASP together with Oracle 8.x.

The system has been tested against real models and the tested result shows that this approach of developing code compliance checking is very successful. Manually checking requires the building plan to be submitted to several departments including FSB, CBPU, PUB for different checking. The whole process of approval will normally take two months or longer. While using the code compliance checking system, building plan will only need to be submitted to the system and user can choose different clauses from different department to check. Checking against all clauses from all departments will only take around 6 hours on a IBM P4 dual-processor PC with 2G memory. The system is implemented by a Singapore local company (NovaCityNets Pte. Ltd) [23] and certified by BCA (Building and Construction Authority) that the checking rules are correct and the checking results are consistant.

By using IFC data as input, the system can communicate with all kinds of CAD system including AutoCAD, ArchiCAD, MagicCAD, etc.

By introducing the idea of CCOM, the system becomes more robust and is able to handle more complex checking. Some checking, such as calculate the travel distance, is very hard to be checked manually and the result is also not accurate. The code compliance checking system can handle this kind of checking very well and the checking result is very accurate compare with the checking result done manually.

By separating the checking rule from checking engine, new rules can be added into the system very easily. Existing rules also can be modified easily.

As a key potion of the code-compliance checking system, the pre-computation of CCOM is tested against 127 unit test data. The following table in figure 8.1 contains some of the testing result.

| Model ID | Model Size (k) | Pre-computation Time (s) | CCOM Data Size (k) |
|---|---|---|---|
| Model-6834B | 640 | 3 | 76248 |
| Model-619A | 378 | 8 | 80948 |
| Model-222C1 | 13122 | 132 | 101060 |
| Model-222C2 | 12986 | 131 | 101860 |
| Model-222C3 | 15139 | 171 | 119996 |
| Model-222C4 | 14157 | 149 | 120956 |
| Model-673 | 634 | 8 | 79640 |
| Model-619B | 387 | 8 | 81612 |
| Model-673F | 697 | 12 | 81080 |

Figure 8.1: *testing result of pre-computation.*

Figure 8.2 shows the relationship between the input data size and the computation time. It is obvious that the computation time will increase when the input data size is increasing. However, it is not linear since the computation time is not only depended on the input data size, but also depended on the input data complexity.

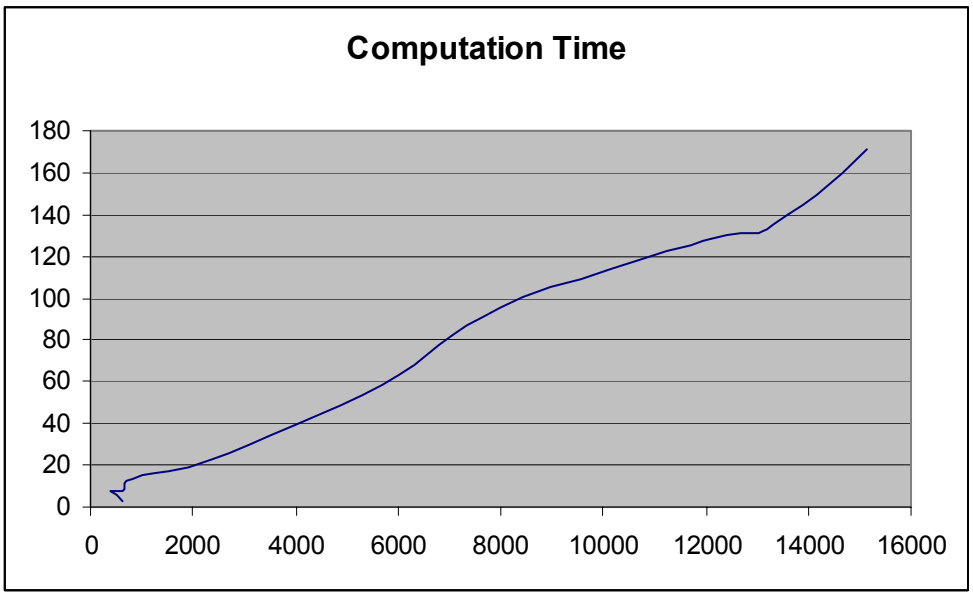



Figure 8.2: *diagram for computation time.*

Figure 8.3 shows the relationship between the input data size and the computed CCOM data size. The overall intention of the CCOM data size is increasing when the input data size is increasing. However, the range of the increasing is not that much. The result shows that the CCOM data size will not to be too big even for a very lag input data.

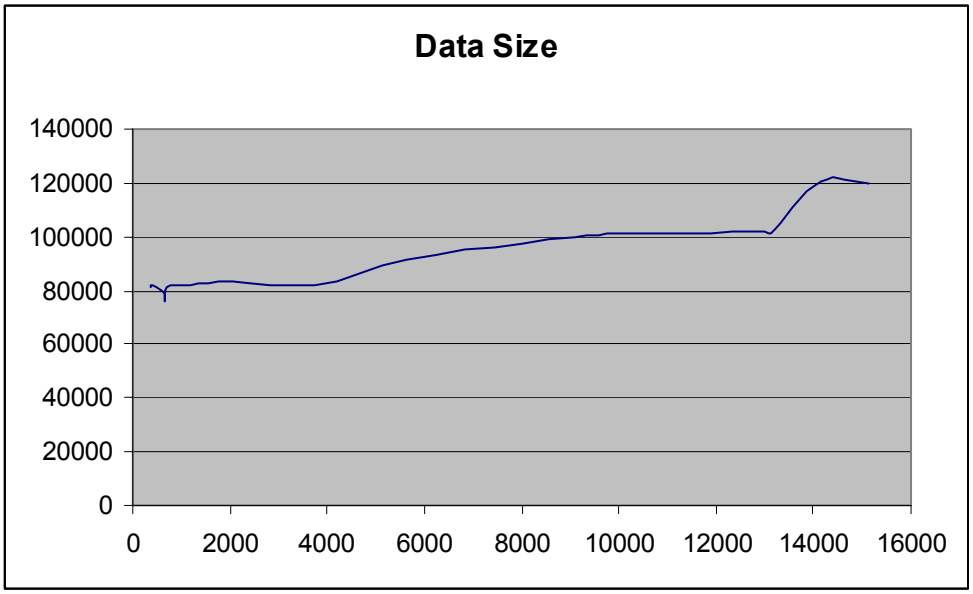



Figure 8.3: *diagram for CCOM data size*

The size of a normal HDB building model, which has around 15 storeys, is around 30,000k. The approximated CCOM data size will be around 165,000k and the approximated pre-computation time will be around 330 seconds. Figure 8.4 shows the checking time using such model to check against different clauses. The server using is IBM P4 dual-processor PC with 2G memories.

| Clause ID | Checking Time (s) |
|---|---|
| 6_1_9 | 21 |
| 6_2_6_3 | 12 |
| 6_2_6_4 | 10 |
| 6_4_1 | 15 |
| 6_4_3 | 32 |
| 6_5_1 | 44 |
| 6_7_3 | 44 |
| 6_12_2_11 | 5 |

Figure 8.4: *checking time for test data*

Compare to existing systems, the new system is more compatable to check against more complex code and regulations. Figure 8.5 shows several categories

of checking that can be handled by the new system. 65% of Singapore codes can be checked using the new system while only 10% can be checked by BP-Expert and 16% can be checked by SMC. The percentage is calculated base on the number of clauses.

| | New system | BP-Expert | SMC | Han's system |
|---|---|---|---|---|
| Property of building element | Y | Y | Y | Y |
| Space relationship | Y | N | Y | N |
| Travel distance | Y | N | N | N |
| Building service connectivity | Y | N | N | N |

Figure 8.5: *checking categories*

# 9. Conclusion

This thesis introduces a novel approach of developing code-compliance checking system. By deriving semantics information from CAD data and introducing building checking model CCOM to store semantics information, the new system can be used to check against more complex codes and regulations than previous systems. As the semantics information has been pre-computed and stored, which is a huge overhead of the computation for previous systems, the online checking time for the same code or regulation also reduced. Also, as CCOM also containing the basic checking logic and geometry computation, the result of code-compliance checking becomes more robust. The checking part of the system is fully automated, i.e., user does not need to input any extra information for the checking.

Of course, there are still a lot of improvement can be done to make the code-compliance checking better. Only the checking part of the system is fully automated but other parts of the system still require manual works done by experts from different domains. In the current approach, the CCOM data model semantics are designed manually by IT experts according to the nature of current AEC codes and regulations. If there are significant changes on codes and regulations, the CCOM data model needs to be revised. Knowledge-based system or expert system can be considered to be used for generating the data model semantics with enough codes and regulations. The checking functions of current approach are also coded by IT programmer with the manually

interpretation of codes and regulations done by AEC expert. To increase the robustness of the system, an automatic checking function generating system can be consider to be implemented to generate checking functions base on codes and regulations.

# 10. Summary

This thesis presents a better approach of developing compliance-checking system.

Original approaches of doing compliance checking are all built up on top of IFC object. Since IFC is introduced to achieve data interoperability, the information provided by IFC is not enough for compliance checking, which requires a lot of semantic information.

In this proposed approach, a new object model CCOM is introduced. CCOM object will read low-level information from IFC object and derive high-level semantics. CCOM model contains two main types of objects, CCOM_Container and CCOM_Element. While CCOM_Element is similar to IFC object, which contains information about single element, the CCOM_Container contains semantic information such as element relationships. CCOM_Container contains more complex data structure to store high-level semantics, such as graph, tree, etc. High-level semantics makes the compliance-checking system to be built much easier and more consistent.

## 11. References

[1] C. S. Han, J. Kunz and K. H. Law, Making Automated Building Code Checking a Reality, Facility Management Journal, September/October, 1997, pp. 22-28.

[2] C. S. Han, J. Kunz and K. H., Law, A Hybrid Prescriptive/Performance Based Approach to Automated Building Code Checking, ASCE J. Computing in Civil Engineering, 12(4):181-194, 1998.

[3] C. S. Han, J. C. Kunz and K. H. Law, Compliance Analysis for Disabled Access, Advances in Digital Government Technology, Human Factors, and Policy. William J. McIver, Jr. and Ahmed K. Elmagarmid (eds) Boston Kluwer, 2002.

[4] International Alliance for Interoperability Industry Foundation Classes, Specifications Volumes 1-4, Washington D. C., 1997.

[5] G. Lau, K. H. Law, and G. Wiederhold, A Framework for Regulation Comparison with Application to Accessibility Codes, Proceedings of the National Conference on Digital Government Research (dg.o2003), Boston, MA, May 18-21, 2003, pp. 251-254.

[6] R. F. Woodbury, A. L. Burrow, R. M. Drogemuller, S. Datta, Code Checking by Representation Comparison, International Journal of Design Computing, 3(1), 2000, pp. 73-89.

[7] California Building Code, California Building Standards Commission, 1998, http://www.bsc.ca.gov/

[8] Singapore Code of Practice on Building Design, Building and Construction Authority, http://www.bca.gov.sg/industry_programmes/buildable_design/legislation/codepractice.html

[9] Q. Z. Yang, C Jason, D Dragan, H. B. Lee, Development of Automated Building Plan Checking Prototypes for AEC Industry, SIMTech Technical Report (MIT/01/016/ITAC)

[10] W. C. Wong, Singapore Automated Plan Checking System, From Dream to Reality, interop AEC + fm 2001, Sydney 29-30 October 2001

[11] L Khemlani, Solibri Model Checker, CADENCE AEC Tech News #87, Nov. 21, 2002

[12] X.J.Tan, F.L.Bian, J.Li, Research on Object Oriented Three Dimensional Data Model

[13] R. Dogan, S. Dogan, M. O. Altan, 3D Visualization and Query Tool for 3D City Model, Geo-Imagery Bridging Continents, XXth ISPRS Congress, 12-23 July 2004 Istanbul, Turkey, Commission 3

[14] Anshuman Razdan , Jeremy Rowe , Mathew Tocheri , Wilson Sweitzer, Adding Semantics to 3D Digital Libraries, International Conference on Asian Digital Libraries (ICADL 2002) , Singapore, December 11-14, 2002

[15] Stephan Nebiker, SUPPORT FOR VISUALISATION AND ANIMATION IN A SCALABLE 3D GIS ENVIRONMENT – MOTIVATION, CONCEPTS AND IMPLEMENTATION, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXIV-5/W10

[16] Sun, M. and Parand, F., Integration of CAD, Product Model and Distributed Building Component Databases, ECPPM 1998

[17] Schiewe, J., Combining geometrical and semantical image information for the improvement of Digital Terrain Models, Buchroithner, M.F. (Ed.): A Decade of Trans-European Remote Sensing cooperation, Proceedings of the 20th EARSEL-Symposium, A. A. Balkema Publishers: 175:180

[18] Rong Xu, Wawan Solihin, Zhiyong Huang, Code Checking and Visualization of an Architecture Design, IEEE Visualization October 10-15, 2004

[19] de Waard, Marcel (1992). Ph.D. Thesis: Computer Aided Conformance Checking: Checking Residential Building Designs Against Building Regulations with the Aid of Computers, The Hague, The Netherlands.

[20] Garrett, J.H., Basten, J., Breslin, J., Andersen, T. (1989) "An object-oriented model for building design and construction," *Proc. Struct.Congress,* ASCE pp. 332-341, New York, NY.

[21] Ito, K., Ueno, Y., Levitt, R.E., Darwiche, A. (1989) "Linking knowledge-based systems to CAD design data with an object-oriented building product model," *Technical Report 17, Center for Integrated Facility Engineering*, Stanford University, Stanford, CA.

[22] Garrett, J.H., Basten, J., Breslin, J., Andersen, T. (1989) "An object-oriented model for building design and construction," *Proc. Struct.Congress,* ASCE pp. 332-341, New York, NY.

[23] Wawan S, Naveed S, Xu R, Lam K P, Beyond Interpretability of building model: a case for code compliance checking, BP-CAD 2004

# Appendixes

Sample of codes and regulations with interpretation

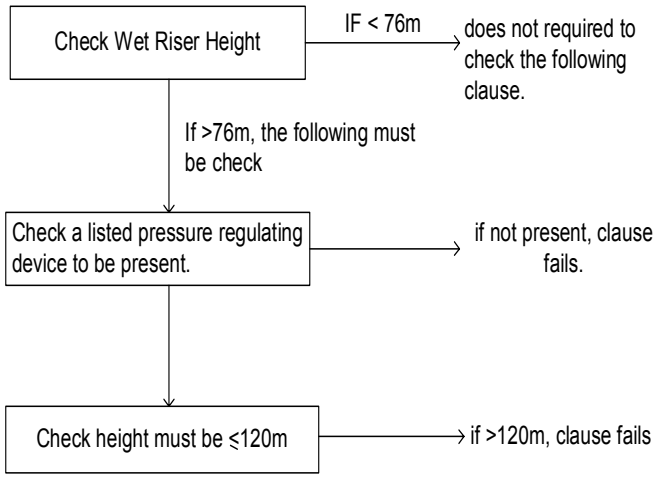
| **CLAUSE**<br>Clause 2.2 Rising Main<br>Clause 2.2.1 Type of System | AUTHOR: EDMUND<br>DATE: 12.01.2001 |
|---|---|

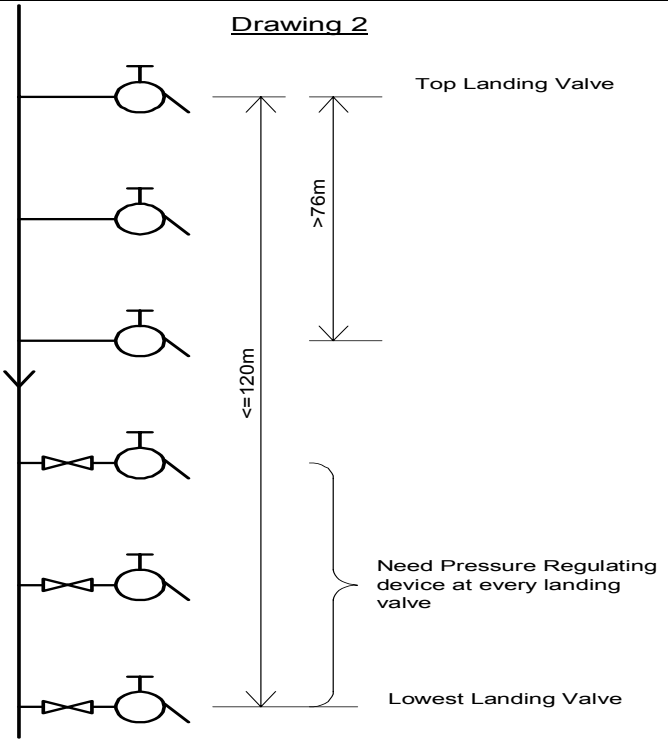
**Code of Practice** (s/n 1, 2)

Clause 2.2.1.2

Wet Rising main zone height exceeding 76 m may be permitted after a listed pressure regulating device, which controls nozzle pressure under both flow and no flow conditions, is installed at outlet and

(a) The maximum zone height shall not exceed 120m.

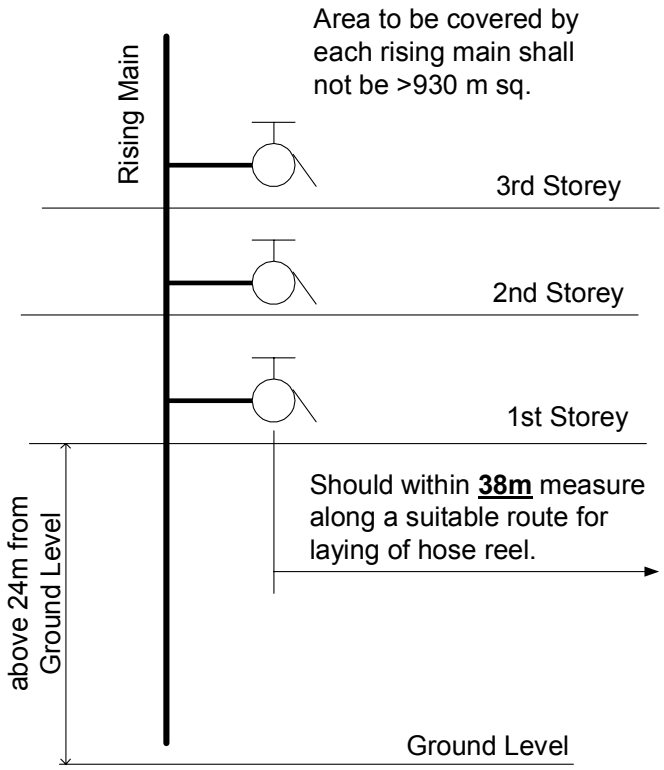
| **Schematic Drawing Representation** | **Interpretation** |
|---|---|
|  <br>**Drawing 1** | This clause check for the wet rising zone height. If the wet riser zone height is >76m, the following must be check:<br>1. there must be a listed pressure regulating device at each outlet position of the riser stack.<br>2. the maximum zone height shall be ≤ 120m.<br><br>Assumption to be made:<br>1. Main zone height is calculated from the lowest landing valve to the highest landing valve from the same stack.<br>2. pressure regulating valve will be at the branch of the landing valve.<br>3. if zone height is between 76m to 120m, check for pressure regulating device in the wet riser stack at every landing valve below 76m.<br><br>* Calculate from the center to center of the pipe. |

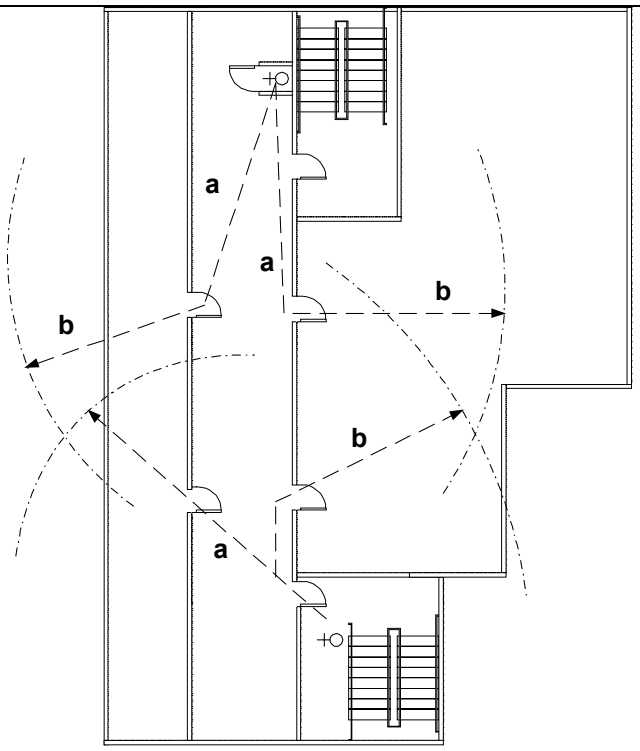

Drawing 2
Top Landing Valve
>76m
<=120m
Need Pressure Regulating device at every landing valve
Lowest Landing Valve

| **CLAUSE**<br>Distance between landing valve & rising main< 38m, if floor is < 24m<br>Part 2.2.2b | AUTHOR: EDMUND<br>DATE: 12.01.2001 |
|---|---|

**Code of Practice** (s/n 4)
Clause 2.2.2b
Number of Rising mains
The number and distribution of rising mains shall be such that:

b) One rising main is provided for one or series of floors higher than 24 m above ground level, with each rising main serving not more than 930 m sq any floor and subject to all parts of the floor to be with in 38 m from landing valve.

| **Schematic Drawing Representation** | **Interpretation** |
|---|---|
|  | This clause the number and distribution of the rising main. For floor higher than 24m above ground:<br>1. at least a rising main in every or series of floor.<br>2. Each Landing valve shall not cover an area > 930m². Each Rising main shall have no more than 2 landing valves.<br>3. The maximum coverage of the landing valve for all part of the floor shall be =< 38m though a suitable route.<br><br>Notes:<br>1. Check that all floors should contain at least 1 landing valve.<br>2. This includes the checking for wet or dry riser.<br>3. All area of the spaces should be cover by the landing valve.<br>4. Suggest using the shortest path algorithm for checking the distance to the landing valve. |




**(i) a + b = < 38m**

**(ii) Overall area to be covered by a single rising main to be within 930m sq.**

5. Centre line of the corridor/doors can be taken to measure the 'suitable route'.

6. Purposed using travel distance calculation to measure all points on the spaces to be covered by the landing valve.

7. For Calculation of the required no. of rising main use:

R = Sum of All spaces area / 930 m³.

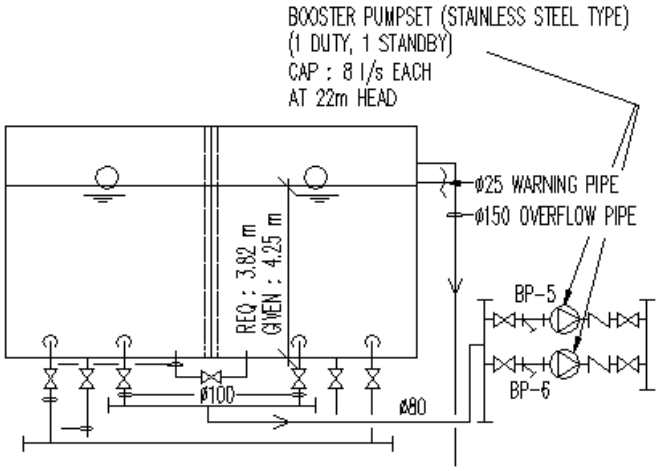
Where R = Required no of landing valve. **Excluding Lift floor area.

| **CLAUSE**<br>Clause 3.1 Water Supply and Pumping arrangements | AUTHOR: EDMUND<br>DATE: 12.01.2001 |
|---|---|

**Code of Practice** (s/n 43)
Clause 3.1.6 Pumps for wet rising mains

Clause 3.1.6.3a
The pumps shall consist of either

(a) 2 fire pumps, at least one of which shall have an independent source of power supply and each shall be capable of providing independently the necessary flow and pressure requirements or

| **Schematic Drawing Representation** | **Interpretation** |
|---|---|
|  <br>**Formula for calculating effective fighting period**<br>$\frac{C1}{v}$ >= 45mins<br>where<br>C1 = Tank Capacity<br>v = Individual Pump Capacity | This clause check for 2 fire pumps to be present at the tank capable to provide independently the necessary flow and pressure requirement.<br><br>When there are 2 pumps are present:<br>- At least one with independent power source.<br>- Each pump capable of providing independent the necessary flow and pressure requirement.<br><br><u>Note:</u><br>QP required to endorse pump specification to meet the requirement of COP as CAD will be unable to check for independent source of supply.<br><br>This clause will be check when 2 fire pumps are detected in the system, the endorsement by the QP that the correct type / requirement of the Pump will be use.<br><br>Sequence of Checking<br>1. Check for number of pump.<br>2. If 2 pumps present check each pump is capable to provide effective fire fighting for a period of 45 mins.<br>3. If 3 pump is present check that the sum of any 2 pump will have the effective capacity for fire fighting for 45 mins. |

| | |
|---|---|
| | **<br>- Do not include Jockey Pump when checking for the number of Pump Set. |